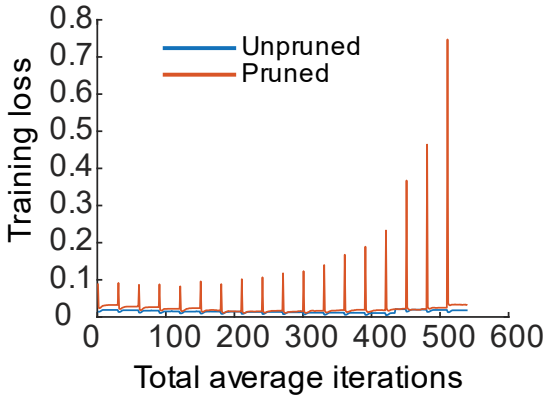

Fig. S1. Training loss across appended training episodes. Each episode corresponds to introduction of a new set of training examples, which differed in their additive noise. For the pruned networks we pruned recurrent connections before retraining. For the unpruned networks we copied the recurrent matrix forward. We then trained to criterion. Because each training episode was a different length, we truncated or NaN padded each episode to make it 30 iterations. We then averaged this across all networks, removing the NaNs.

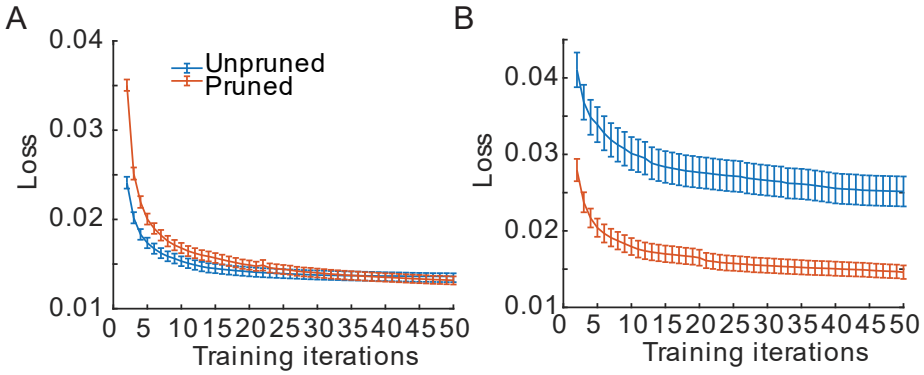

Fig. S2. Training Loss of pruned and unpruned networks in two different learning situations. **A.** Training loss when training with a randomly initialized but unpruned network, or a randomly initialized and then pruned network. Note the first iteration has been removed because there is a large drop on the first iteration (see Fig. S1). Data are for 200 pruned and 200 unpruned networks. **B.** Training pruned (70%) and corresponding unpruned (trained in parallel) networks to learn to do delayed match to sample with new cues, in addition to the previously learned cues. The new cues were operationalized by using 0/1 inputs on new connections that had been clamped to 0 during baseline training. Data are for 100 networks.

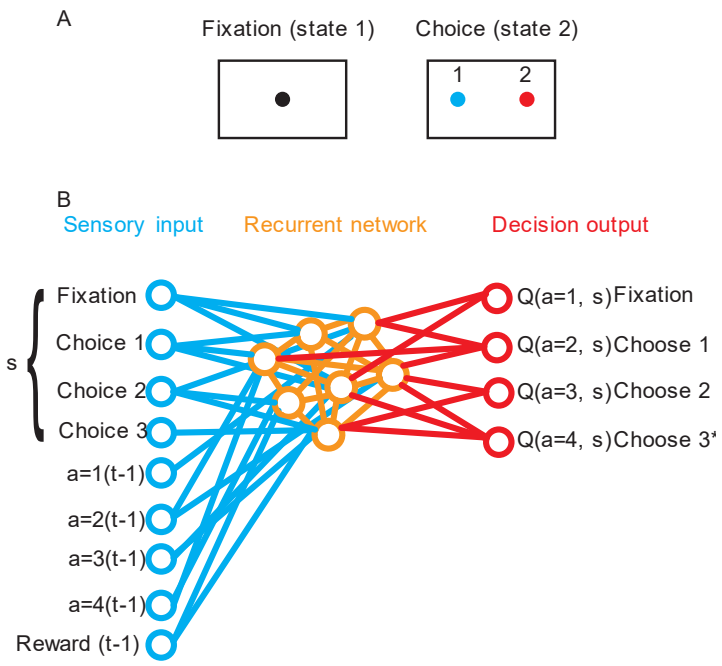

Fig. S3. Reinforcement learning task and network trained to carry out task. **A**. We trained a network on a standard RL task with two states. In the fixation state, indicated by the presence of a fixation point on the screen, the network was trained to fixate the central target. Choice of the fixation target resulted in a reward of 0 and choice of any other option resulted in a reward of -1. In the choice state, two choice options were presented. In each 30 trial block, one of the choice options was more frequently rewarded when chosen. The network had to learn to choose that option in each block, when the choice cues were presented. For most simulations only choice options 1 and 2 were valid. For the exploration training, choice options 1 and 3 were presented, and the network had to learn to select either option 1 or 3 during the choice phase. The fixation target could be selected during the choice state, and a target choice (effectively a screen location) could be selected during the fixation state. However, these were always incorrect choices. **B**. Schematic diagram of network inputs and outputs.

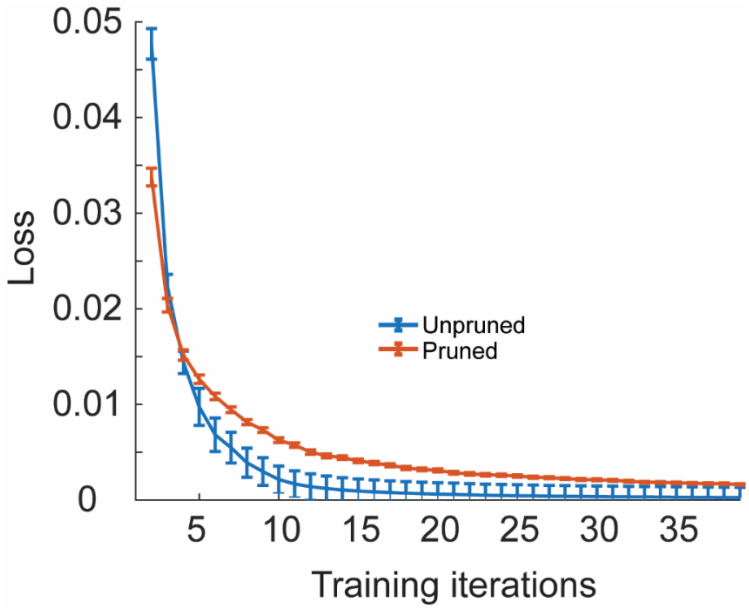

Fig. S4. Training loss when learning a novel option in the RL task. Lines are medians and not means due to some outliers. Error bars are SEM, with N = 100. We have eliminated two unpruned networks that failed to converge and had high training loss (> 10) that did not decay over time. In addition, there were 12 unpruned networks that had very high loss over the first 4 training iterations, that then dropped. Therefore, we removed these 12 from calculation of the SEM, but only for the first 4 iterations. They were included in calculation of the medians.

Supplemental methods

For the delayed match to sample (DMS) task, the inputs, $u(k)$ for each task condition were given by a 6 x K matrix, with K = 58, and *k* indexing the column. Nonzero entries in the first row indicated presentation of cue 1, which could be on the right (1) or left (-1). Nonzero entries in the second row indicated presentation of cue 2 which was on the right (1) or left (-1). When we retrained the network on new cues (Fig. S2), we then used rows 4 and 5 similarly, to represent the new cues. The sixth row indicated trial state. It was set to 1 for the first 10 time points. It also provided a go cue that indicated when a response should be produced (1) or not (0). We also used an additional input, that we do not show, to provide a bias term to the recurrent layer units. This input was always set to 1 (not depicted in Fig. 1B).

A training batch, which was held constant for a training episode, was composed of 20 trials, with 5 trials of each of the four conditions. The desired output, $\hat{y}(k)$, for the delayed match to sample task was 0 at all points, except when the go cue was given, at which point the correct output was 1 for a match and -1 for a non-match. We added a small amount of noise (0 mean Gaussian noise, $\eta(k)$, with a standard deviation of 0.1) to each time point of $\hat{y}(k)$ for all trials. The noise was held constant, i.e. frozen, for a training episode, but differed across episodes. Training for an episode was stopped when the loss was less than 0.0105 and the change in the loss was less than 5 x 10^{-7} or the number of CG

iterations exceeded 100. One training iteration was a full CG update of all conjugate dimensions across all trials in a batch. Note that per Fig. 5C, most networks produced accurate results in the absence of distractors. Distracting probe cues used in some of the simulations, were simulated by inputting an extra cue in the first row during the delay period (Fig. 1A, B). Thus, probes were always delivered in the same input dimension in which cue 1 was delivered. The probes were always on the opposite side from cue 1, and thus a response to the probe would lead to an error. Probes were delivered at a series of time points after cue 1, and at a series of strengths. Networks were never trained on probes. Probes were only delivered when network performance was assessed.

For the RL task, the inputs, $u(k)$for each task condition were given by a 10 x K matrix, with K = 60, and *k* indexing the column. The first row was set to 1 when the fixation cue was presented. The second through fourth rows were set to 1 when choice options 1-3 were present on the screen. For most results we only used choice options 1 and 2. However, when we retrained the network to "explore" the novel option, we used choice options 1 and 3. Only the fixation cue or the choice cues were presented at each time. The fifth through eighth rows were one-hot inputs that indicated the previous choice, and the ninth row indicated the reward outcome for the previous choice. The last row (not depicted in Fig. S3) was set to 1 and used as a bias input to the recurrent units. The RL task was a two-state RL problem, with state 1 given by the presence of the fixation point, and state 2 given by the presence of the two choice targets. The choice and reward feedback inputs to the network, therefore, arrived when the network was making the next choice, always in the other state, as we did not use an inter-trial interval. Each block had 30 trials, where a trial was one fixation choice and one option choice.

The target function, $\hat{y}(k)$ was a 4-element vector giving the Q-value for each of the 4 choices (1). Choice 1 was fixation, choice 2 was target option 1, choice 3 was target option 2, and choice 4 was target option 3. The output of the network was passed through a soft-max to generate choice probabilities:

$$d_i(k) = \frac{\exp(\beta y_i(k))}{\sum_{j=1:4} \exp(\beta y_j(k))} \quad (1)$$

Where $d_i(k)$ is the choice probability for choice *i*, and $y_i(k)$ is element *i* from the network output vector, $y(k)$. The variable β is an inverse temperature parameter set to 3 for all simulations. Choices were sampled from the choice probability distribution, $d(k)$, and further followed an ε greedy policy. The variable ε was initially set to 0.15, and it decayed linearly over 100 full conjugate training iterations during initial training and when learning a new option. During the training and pruning procedure, the variable ε was initially set to 0.15, and decayed linearly over 10 conjugate training iterations.

The RL network was trained on simulated data from a 2-armed bandit task. In the task, when the fixation point was presented, the network received a reward of 0 if fixation was chosen, and a reward of minus 1 (-1) if any other option was chosen. For each block of simulated trials, either option 1 or option 2 was set as the more highly rewarded option, and this was balanced across blocks during training. Option 3 was used for a separate set of simulations, discussed below. When the more highly rewarded option was chosen in a block of trials, it was rewarded with a probability of 0.8. When the other option was chosen it was rewarded with a probability of 0.2. The target function, $\hat{y}(k)$, was generated through Q-learning:

$$Q_{k+1}(s,a) = Q_k(s,a) + \sigma(R(k) - Q_k(s,a)) \quad (2)$$

The variable σ was set to 0.7 for all simulations. The state, s, was 1 for fixation, and 2 for choice periods. The variable a defined the 4 possible choices, as described above. We then set $\hat{y}(k) = Q_k(s,a) + \eta(k)$ for the chosen action and state. The variable, $\eta(k)$, was zero mean Gaussian noise. For the original training, this variable was set to 0. However, to simulate the noisy choice behavior of young agents, we added noise to the target function during the pruning and retraining. We trained pruned and unpruned networks with the same noise level. We also used 3 noise levels with standard deviations of 0.01, 0.1 and 1. Effects of noise were small at 0.01. Noise values of 0.1 and 1 decreased the accuracy of the network, similar to what is seen in experimental work (2). Most results, therefore, are shown for a noise value of 0.1. Note that the REINFORCE algorithm can also be used to train RL networks (3). However, the conjugate gradient algorithm requires stable errors across a full set of updates. The stochastic nature of trial outcomes in the REINFORCE algorithm does not allow for this. Therefore, we chose our method to have a stable target function.

During training, the network was run on a block of trials, and the output of the network, $y(k)$ was used to define choice probabilities using equation 5. These were then used to select actions for each trial. The actions and their associated rewards were fed back into the network. These actions and rewards were also used to update Q-values in parallel using equation 6. In addition, we ran 24 blocks of trials, balanced for whether option 1 or option 2 was the best option. These 24 blocks defined a single training episode. After running the recurrent network agent on the 24 blocks, the Q-values, with added noise, were passed back to the conjugate gradient algorithm, and the network weights were updated through 20 full conjugate iterations. Residuals for all 24 blocks were pooled in the BPPT updates. This sequence was then repeated. Thus, during training, the network was used to generate choices probabilities, these were sampled, and Q-values were generated in parallel. Training for an episode was stopped when the loss was less than 0.01 or the change in loss was less than 10^{-8} or the number of CG iterations exceeded 40. For presentation of results, performance was assessed by calculating the Loss (equation 4) which was used for network training and also by comparing the frequency with which the network selected the same option as the Q-learning algorithm. Note that even when the network predicts Q-values highly accurately, the network and the Q-learning algorithm may select different options because of the soft-max choice rule.

For both networks, pruning was done incrementally, with each round of pruning followed by retraining. Additionally, we trained a set of pruned and unpruned networks in parallel, and both had the same number of training episodes. We started by first training unpruned networks until they were fully trained. We then pruned 10% of the weakest recurrent connections in the matrix *A*, and retrained the network until it was fully trained. We identified the weakest connections by sorting on the absolute value of the connection strength. Therefore, strong positive and negative connections were kept, and weak positive or negative connections were pruned (i.e. permanently set to 0). For unpruned networks, we copied the weight matrix forward, and again trained the network on new training examples, equivalent to the pruned network. We then pruned an additional 5% (20% for RL) of the weakest connections and retrained the network. We continued this until we had pruned 95% (90% for RL) of the connections. We examined performance as a function of the fraction of pruning for both networks. For the DMS network we also compared this incremental approach, to an approach in which we randomly initialized the network, then pruned a large fraction of connections, and then trained the network. For the RL network we examined the ability of the trained and pruned network to learn a new choice option, relative to a trained but unpruned network. It is not clear whether cortical pruning is gradual or

abrupt, as studies that work with post-mortem tissue are necessarily cross sectional (4). However, the cross-sectional work suggests that pruning follows an exponential time course and takes place over the course of at least 10 years in humans (5). This data is consistent with human imaging work that looks at grey matter volume (6), which may reflect synapse density. Thus, pruning and retraining in biology, in cortical networks, is likely a gradual process, like what we have done here.

1. C. D. Marton, S. R. Schultz, B. B. Averbeck, Learning to select actions shapes recurrent dynamics in the corticostriatal system. *Neural Netw* **132**, 375-393 (2020).
2. K. Nussenbaum, C. A. Hartley, Reinforcement learning across development: What insights can we draw from a decade of research? *Dev Cogn Neurosci* **40**, 100733 (2019).
3. R. J. Williams, Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach Learn* **8**, 229-256 (1992).
4. G. N. Elston, T. Oga, I. Fujita, Spinogenesis and pruning scales across functional hierarchies. *J Neurosci* **29**, 3271-3275 (2009).
5. Z. Petanjek *et al.*, Extraordinary neoteny of synaptic spines in the human prefrontal cortex. *Proc Natl Acad Sci U S A* **108**, 13281-13286 (2011).
6. J. N. Giedd, Structural magnetic resonance imaging of the adolescent brain. *Ann N Y Acad Sci* **1021**, 77-85 (2004).